An Infrastructure for Network Computing with Java Applets

Arash Baratloo, Mehmet Karaul, Holger Karl, and Zvi M. Kedem
(baratloo|karaul|karl|kedem)@cs.nyu.edu
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University

Abstract

Java in combination with Web browsers' abilities to load and execute untrusted Java applets in a secure fashion has made computing over the Web a possibility. Now the challenge is to fully utilize this potential, given the limitations imposed by browsers. This paper presents *Knitting-Factory*, an infrastructure to facilitate Web-based computing, which addresses this challenge. It supports building distributed applications, specifically those consisting of Java applets executing in browsers. It is composed of (1) a distributed name service to assist users in locating other participants of a distributed computation via standard browsers, (2) an embedded class server to eliminate the need for external HTTP servers for serving applet code, and (3) a technique for direct applet-to-applet communication.

In this paper, we describe the design and implementation of KnittingFactory and demonstrate its benefits by applying it to three distinct areas of Web-based computing. First, we apply our distributed name service to a client/server architecture to enable RMI clients to locate servers on unknown hosts. Second, we use the embedded class server to extend the capability of *Charlotte*, a parallel computing environment. And finally, we build a collaborative application using our direct applet-to-applet communication technique which does not require a forwarding agent.

1 Introduction

The World Wide Web has the potential of integrating remote and heterogeneous computers into a single global computing resource for parallel and collaborative work. While parallel computing on workstation clusters is common practice with systems such as PVM [1] and MPI [2], this is not the case for computing over the Web. Similarly, while collaborative work on intranets is supported by systems like CORBA [3] and DCOM [4], this does not hold true for the Web. Some of the obstacles common to both Web-based parallel computing and collaborative work are the heterogeneity of the participating systems, difficulties in administering distributed applications, security concerns of users, and often high communication delays.

The combination of the Java programming language together with Java-capable Web browsers has successfully addressed some of these problems. Java's platform independence solves the problem of heterogeneity. The growing number of Java capable browsers able to seamlessly load applets from remote sites reduces administration difficulties.[1] The applet security model, which in most parts enables browsers to execute untrusted applets in a trusted environment alleviates some of the users' security concerns. It is because of these reasons that Java applets became a good candidate for building Web-based systems, and browsers the perfect candidate to seamlessly bring distributed computing to every-day users.

In this paper, we focus on applications composed of Java applets and designed to run within Web browsers. As a result of limitations put on applets, the architectures of many existing systems have the following characteristics: (1) a stand-alone application running on the same host as an HTTP server; (2) users on the Internet joining

[1] Consistent with Java terminology, we use the term *applet* to refer to a Java program which is typically loaded form the Internet and executed inside a browser.

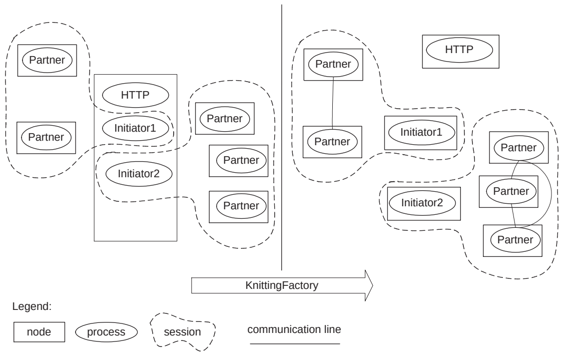

Figure 1: With KnittingFactory, applications can execute anywhere on the network; browsers are able to locate the application; and after a *bootstrapping* procedure, direct applet-to-applet communication is possible.

a stand-alone application by loading an applet using an a priori known URL and executing it within a browser; and (3) each applet creating a direct network connection to its associated application, which is responsible to propagate information from one applet to another (see the left side of Figure 1). In an ideal situation, (1) the applications should not be tied to an external HTTP server and should be allowed to execute anywhere on a network; (2) users should not be required to have a priori knowledge of the URL and should be able to locate these applications in a seamless fashion via browsers; and (3) collaborative applets should be able to directly exchange information. The ideal scenario is depicted in the right side of Figure 1. KnittingFactory is an infrastructure for building such systems and provides three integrated services:

- *KF Directory Service*, a distributed name service to assist users in finding networked applications on unknown hosts. *KF Directory Service* makes it possible for the user to find such applications on the Internet by performing a lookup at any site offering a *KF Directory Service*. Since we are mostly concerned with applications which register and deregister frequently, the challenge is to provide this functionality in a decentralized fashion, but more importantly, such that it works with Web browsers.

- *KF Class Server*, an embedded class server to eliminate the need of external HTTP servers. A user who may wish to initiate a distributed application might not have access to a host running an HTTP server and it may be inconvenient or impossible to install such a server on the local host. *KF Class Server* provides a light-weight solution to allow users to initiate a distributed effort on any host.

- *KF Applet* for direct applet-to-applet communication. The host-of-origin policy imposed by browsers prevents direct network connections between applets. KnittingFactory is unique in providing an infrastructure for applets of the same distributed session, executing in multiple browsers on the Internet, to communicate and exchange information *directly*.

The rest of this paper is organized as follows. Next, we give an overview of related projects to highlight how KnittingFactory can assist in overcoming some difficulties associated with applets. Section 3 describes the philosophy and design of KnittingFactory. Sections 4, 5, and 6 describe the implementation and example applications of the respective services. A more detailed description of the implementation can be found in [5]. We conclude by discussing the security implications of KnittingFactory and by presenting the status and future directions of this project.

2 Related Work

There is a growing body of work on how to best utilize the emergence of new Java-based technologies. This work can be loosely categorized based on their emphasis: those that focus on parallel programming environments, or *metacomputing on the Web*, and those that focus on the infrastructure for *collaborative applications on the Web*.

Metacomputing on the Web: The projects which focus on Java-based parallel computing include ATLAS [6], Charlotte [7], JPVM [8], JavaParty [9], Javelin [10], ParaWeb [11], and WebFlow [12]. Similarly, there have been proposals for Web-enabled virtual machines as a base for High Performance Computing and Communication (HPCC) platform [13], to use Java in an SPMD programming model [14], and to extend Java with global pointers and remote service request mechanisms from the Nexus communication library [15]. Unfortunately, all but Charlotte and Javelin fail to take advantage of Web browsers in bringing distributed computing to every-day users.

JPVM provides a message passing interface to Java stand-alone applications, but not applets. Projects such

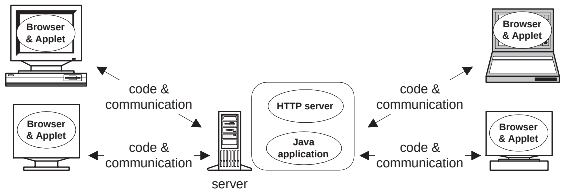

Figure 2: Architecture of a typical Web-based parallel or collaborative computing system.

as JavaParty and [14] require running a Java process to act as a runtime environment (in the case of JavaParty, this process is called the *LocalJP*). ParaWeb includes a distributed implementation of the Java Virtual Machine. ATLAS programs are load-balanced using work-stealing scheduling but rely on native code which eliminates the secure program execution guarantees.

Charlotte and Javelin were specifically designed for parallel programming over the Web. By leveraging the ability of browsers to download and execute remote applets, they provide the means for any user, anywhere on the Internet, using any Java-capable browser to participate in a parallel computation. The essence of their system architecture is shown in Figure 2. A single host, the server in the diagram, runs a Java stand-alone application and an HTTP server. Other hosts load applets which reside on the server and execute them within a browser's secure environment. It is important to note that all applet-to-applet communication is routed through the stand-alone application running on the server.

There are two features common to the design of Charlotte and Javelin. First, they require a host running an HTTP server in addition to a stand-alone Java application. In Charlotte, the stand-alone application is called the *manager* and there is one manager per parallel program; in Javelin, it is called the *broker* and a broker may serve multiple programs at a time. The role of the stand-alone application is to distribute work among browsers and to act as a message forwarding agent for applet-to-applet communications. A single forwarding agent has the potential of becoming a bottleneck. The second commonality is that users need to have *a priori* knowledge of the host running the stand-alone application.

Collaborative applications on the Web: The second category of projects focuses on software infrastructures for collaborative applications. Examples of such applications are distributed whiteboards, calendars, and editors where multiple users collaborate towards one goal. High-level software systems which support collaborative Java applications include the Caltech Infosphere project [16], Jada/PageSpace [17], Java Collaborator Toolset [18], JavaSpaces [19], and TANGO [20].

Jada and JavaSpaces integrate the Linda concept with Java. Where Jada uses a Linda-like tuple model, JavaSpaces utilizes object hierarchies for tuple matching. PageSpace builds higher-level functionalities on top of Jada. Infosphere proposes channel-coupled agents to build flexible ways of collaborative interactions. These systems utilize Java for building stand-alone applications and do not take advantage of browsers. For example, while users interact with a PageSpace application in a browser, the actual collaboration is provided by a set of processes which must be started beforehand.

TANGO provides functionalities such as session management, user authentication, event logging, and communication between collaborative users. The main two elements of TANGO's architecture are a *local daemon* and a *central server*. The local daemon is implemented as a Netscape plug-in to circumvent restrictions put on applets, and hence, breaks the security guarantees.

Java Collaborator Toolset provides a replacement for Java's standard Abstract Window Toolkit (AWT). The replacement package propagates AWT events to all collaborating agents. However, this is done through a central *event distributor*, similar in design as depicted in Figure 2.

While the functionality of these metacomputing and collaborative systems transcends KnittingFactory's goals, KnittingFactory can provide a flexible infrastructure for them. First, *KF Directory Service* can provide the mechanism for users' browsers to search the Web for work. Second, *KF Class Server* can be used to run initiators on machines other than a common HTTP server machine. And finally, *KF Applet* can support direct applet-to-applet communication avoiding a potential for a bottleneck.

3 Design

The design of KnittingFactory has the following goals:

- *Maintaining ease of use:* KnittingFactory should work with standard browsers and comply to Java standards. In particular, users should not be burdened with additional administrative efforts such as locally installing software packages.

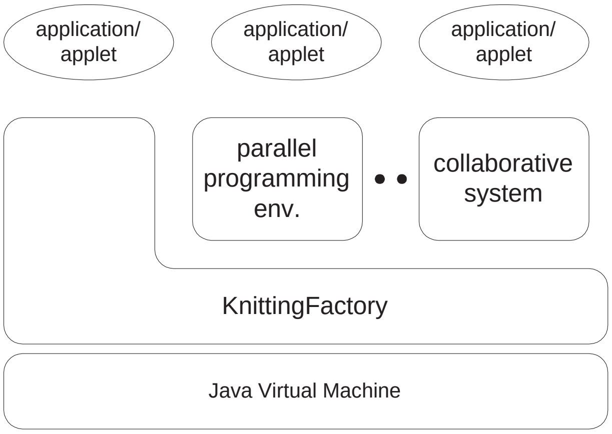

Figure 3: Layered design of KnittingFactory. It supports distributed applications, as well as functioning as an infrastructure for other higher-level environments.

- *Maintaining secure environment:* The system security should not be jeopardized. This means that both native code and browser plug-ins should be avoided. In hindsight, this is an unresolved issue and will be further discussed in Section 7.
- *Scalability:* Since KnittingFactory targets computing over the World Wide Web, scalability is a consideration.

To take full advantage of today's technology, parts of the application designed to execute on remote machines are generally implemented as applets. The code to execute on a user's local machine is implemented as a stand-alone Java application which is not under scrutiny of the applet security model. In this paper, we call the combination of a stand-alone application and applets a *distributed computation*, a particular invocation of this a *session*, an invocation of the stand-alone application an *initiator*, and applets belonging to the same session *partners*. This scenario is the setting for KnittingFactory.

While many of the systems surveyed in the previous section provide high-level functionality to programmers and users, KnittingFactory is concerned with providing the infrastructure of such systems. Figure 3 shows the layered design of KnittingFactory. In Section 5 we show how it can be used to support parallel programming environments, and as an example, we extend Charlotte's capabilities. In Section 6 we show how it can directly support collaborative applications.

The design of KnittingFactory's three services, *KF Directory Service*, *KF Class Server*, and *KF Applet* is outlined below.

3.1 Registration and Lookup — *KF Directory Service*

Distributed applications are generally easier to build and use when a lookup- or directory-service is available. While there are many classical solutions for this, for example CORBA name servers, our focus is to provide a light-weight solution for the Web. In particular, we aim (1) to allow lookups from within a browser, (2) to use the Web-infrastructure as much as possible, (3) not to introduce new daemon processes for performing this service, and (4) to accommodate highly dynamic registration and deregistration of entries to support short-lived sessions.

A popular directory service is the Lightweight Directory Access Protocol (LDAP) [21]. It specifies both a global data model and a protocol for clients and servers to access and maintain this directory. The servers perform search requests on clients' behalf. Our goal is to accommodate a large number of client requests for highly dynamic directory entries. The dynamics of the directories makes replication of information among distributed directory servers unattractive because of frequent invalidation. But without replication, the servers would have to perform considerable work to service the requests for a large number of clients. Therefore it is not feasible to perform the search within the servers, and consequently, LDAP-based directories are ill-suited for this scenario. We chose to pursue a solution which executes the search within a client's browser.

Using dynamically loaded applets to realize a directory service is also not feasible. Consider an applet sent from host A and executing within a browser on host B. The applet can establish a network connection only to host A. If the user at B is interested in performing a directory service lookup, it is either the case that (1) host A acts as a single central directory service for this and all other sites, or (2) host A performs the search on behalf of B, or (3) the directory information is replicated and synchronized on multiple sites—neither option is acceptable.

In KnittingFactory, we build upon existing HTTP servers to act as *directory servers*. Such a directory (1) stores information about sessions looking for prospective partners (such as the URL of the initiator, comments, *etc.*), and (2) maintains a list of other *KF Directory Services*, which can be interpreted as the edges of a directed graph of *KF Directory Services*. Adding and removing of both, sessions and directories can be done using standard protocols (HTTP Post and CGI scripts).

Our technique to support lookup operations is unique. A lookup can be initiated at any server which offers a *KF Directory Service*. By including a JavaScript program along with a session and directory list in a single HTML page retrieved from such a server, a browser can inspect this information and, if a matching session is found, proceed to the respective URL, otherwise use the directory list to search recursively at another site. By passing along state information as part of the URL of the next directory, search strategies like breadth-first search can be implemented.

3.2 Arbitrary Origin of Applets — *KF Class Server*

In a typical distributed Java application, applets are loaded from an HTTP server and connect to a stand-alone application, the initiator (see Figure 2). Among others, this is the case for applications written in Charlotte, Javelin, and the Java Collaborator Toolset. The Java security model dictates that the initiator has to run on the same host as the HTTP server. Ideally, however, initiators should be able to execute anywhere on a network.

KnittingFactory addresses this problem by providing a light-weight HTTP server. *KF Class Server* implements the essential functionalities needed to serve applets, and is designed to be embedded into any Java application. As a result, it is the application itself which dynamically serves classes to browsers upon request. Although quite simple, this makes it possible for initiators to run on any host connected to the network.

3.3 Direct Applet-to-Applet Communication — *KF Applet*

Browsers impose their own restrictions, specifically the *host-of-origin* policy, to safely execute a piece of code loaded over the network. In short, browsers disallow applets from establishing a network connection except to the host where they were loaded from. This has hampered the "collaborativeness" of applets. Typical solutions to allow applets to exchange information have been to either rely on *untrusted native code* or to use *a single forwarding agent*. Untrusted code has been used as native-methods [6] and browser plug-ins [20]. However, this approach breaks the security guarantees provided by Java and introduces additional administrative overhead otherwise not needed. More importantly, it prevents off-the-shelf browsers from participating in a distributed computation. The use of a single agent to forward messages among applets as used in [10, 18, 17] has drawbacks: it introduces a single point of failure, limits scalability, and might raise privacy concerns when messages should not be subject to inspection by a forwarding agent.

JDK 1.1 introduced signed applets and JAR files using a certificate [22, 23]. In JDK 1.1, once a signature is recognized and trusted, a signed applet is treated as trusted local code. For future releases [23], Java promises permission specifications and fine-grain access control. The Java runtime is to maintain a mapping from a *code source* (i.e., the URL location and a corresponding certificate) to a *protection domain* and then to user-defined permissions. This way, a user can define the permission of executing code and selectively override the host-of-origin policy by trusting certain code-providers. But the process of contacting a certifying authority for a certificate is a tedious process for most casual application developers. KnittingFactory provides a flexible solution without burdening the user with the problematic choice of where to put trust, and without burdening the developer with obtaining a certificate.

KF Applets use Remote Method Invocation (RMI) [24] of Java 1.1 to realize direct applet-to-applet communication. RMI is based on a client/server architecture; once a client obtains a reference to a remote server object it can invoke server methods. In our design, each *KF Applet* creates such a remotely accessible object and passes its reference to the initiator. In the simplest case, the initiator redistributes the references back to all other applets. Thus, every *KF Applet* belonging to the same session can invoke others' methods and thereby communicate arbitrary data through method arguments.

4 Implementation of *KF Directory Service*

KF Directory Service is implemented by a set of HTTP servers acting as directory servers, and a single HTML file at each server containing a table of directory entries and (as will be described below) a JavaScript program.

While every HTML file contains the same JavaScript program, their directory entries may differ. A directory entry is a tuple containing a string (session name), a URL address (initiator of the session), and optional fields for category, password, and comments. The category field is used to distinguish between entries representing sessions and other directory servers. Adding and removing an entry is performed by processes sending HTTP Post messages to HTTP servers resulting in execution of appropriate CGI scripts.

A *lookup operation* consists of loading an initial HTML page and executing the embedded JavaScript program with the request as a parameter. This program analyzes the entries in its own page. If no match is found, a new URL is constructed and the browser is instructed to load the new address. The new URL contains the address of another directory server augmented with state information containing a list of hosts visited so far, and a list of hosts to visit next which is included in the tag part of the new URL. This state information allows the implementation of different search strategies, e.g., breadth-first search to find sessions on nearby hosts first. Our implementation allows the search to be performed by a JavaScript program without breaking the security restrictions imposed by Web browsers. Under the *tainting* [25] security model, a JavaScript program is not allowed to inspect pages coming from another site than itself. Once the JavaScript program finds an entry matching the request, the browser is instructed to follow the corresponding link. In our case, this will download an HTML page with an embedded applet which constitutes a partner of a distributed computation.

Note that this is not a search engine: unlike centralized search engines, the *KF Directory Service* is truly decentralized. The only services required from the HTTP server are sending the HTML file and running the CGI scripts to add and remove entries. The actual search takes place in users' browsers.

4.1 *KF Directory Service* and Java RMI

While *KF Directory Service* was initially designed to work within browsers, in this section we will consider an example how its services can be used in other contexts. In particular, we will look at the client/server architecture of Java Remote Method Invocation (RMI).

In Java RMI terminology, a *registry* is a remote object which provides basic name server functionality. The `rmiregistry` provided in JDK 1.1 is a shell script which invokes `RegistryImpl`, an implementation of such a registry. Two methods provided by the registry are of special interest to us: `bind()` and `lookup()` for registering and locating a server object, respectively. The existing RMI registry successfully binds a server object *only if it is local to its machine*. In the absence of a network-wide directory service, this means either that client applications must have *a priori* knowledge of the host running each server they might contact, or that all servers must run on a single well-known host.

KnittingFactory overcomes this restriction by providing a directory service to RMI registries. We are in the process of implementing a wrapper class around `RegistryImpl`. In addition of passing `bind` operations to the RMI registry, our wrapper class makes a corresponding entry in a *KF Directory Service* under a specified service name. Our `lookup` operation uses *KF Directory Service* to first search the network for the host running an RMI registry with the appropriate entry and then contacts that registry for a server's remote reference. Thus, with the help of *KF Directory Service*, RMI servers can run and execute anywhere on the network, and clients can transparently get servers' remote references without knowing which hosts to search.

5 Implementation of *KF Class Server*

KF Class Server provides the core functionalities needed to allow initiators to execute anywhere on a network. In particular, it serves an initial HTML page containing an applet tag, which implicitly results in additional requests for the applet classes which are served as needed.

KF Class Server is implemented by the `KF_Server` class. Any stand-alone application, such as an initiator, can instantiate a `KF_Server` object to use its services. `KF_Server`'s constructor takes the name of the applet class (i.e., partner) to be included in the initial HTML page, creates a server socket, and listens for HTTP requests. An HTTP request for a Java applet class results in asking the underlying Java Virtual Machine for the requested class and returning the corresponding byte-code.

KnittingFactory also provides a specialized implementation of `KF_Server` which integrates *KF Directory Service*. In what follows, this implementation is used as an underlying layer for a parallel programming environment.

5.1 *KF Class Server* and Parallel Programming Systems

We chose the *Charlotte* [7] system to evaluate the infrastructure provided by KnittingFactory. Charlotte is a parallel programming environment which provides features analogous to distributed threads and shared memory abstractions. The two main components in a Charlotte program are: a *manager* (i.e., initiator) and one or more *workers* (i.e., partners). The manager process creates an entry in a well-known HTML page at initialization. Users can load and execute the worker code by directing their browsers to this URL.

Charlotte requires that a manager runs on a host with an HTTP server to enable communication between manager and workers. With only six lines of code, KnittingFactory permits such a manager to be started on any host and to register itself with the *KF Directory Service*, allowing clients to find Charlotte applications easily. If only one machine with an HTTP server is available (a common setup), multiple Charlotte managers no longer need to run on this single machine, but they can be started on arbitrary machines (see Figure 1).

6 Implementation of *KF Applet*

This section addresses the issue of how to facilitate direct communication among multiple applets belonging to the same session and why this is beneficial. The security concerns regarding such inter-applet communication will be discussed in Section 7.

Our approach is based on the RMI interface of Java 1.1. `KF_Registry` implements the functionality of an RMI registry. A `KF_Registry` object is a remote object which registers with its own registry during startup of the initiator (see Figure 4, step 1). A joining applet retrieves a remote reference to this object (e.g., after finding the initiator through *KF Directory Service*) from the RMI registry[2] (step 2), and invokes a remote method of `KF_Registry` to inform the registry of its presence, passing along a reference to itself (step 3).[3] `KF_Registry` stores this reference and returns references to already registered *KF Applet*s to the joining applet (step 4). Finally, the new applet registers itself with

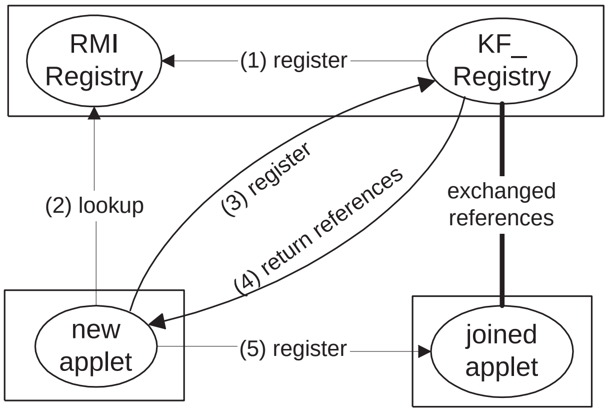

Figure 4: Registration protocol for applet-to-applet communication.

all other applets, passing alone a reference to itself (step 5). After this, all applets know about each other and can invoke remote methods among each other.

After this bootstrapping phase, the `KF_Registry` is no longer needed—applets can use their mutually known references to invoke RMI calls at each other and thereby communicate directly without using the initiator in any way. This removes the initiator as a potential bottleneck since it is only involved during the bootstrapping phase, and the latencies experienced by users of a collaborative application should be much lower compared to an approach using a forwarding agent. For example, consider two participants in Europe cooperating via an applet originating from a server in North America. KnittingFactory provides the `KF_Applet` class as a convenience wrapper which hides the details and allows remote method invocation at other participating applets; the initiator has only to instantiate an object of class `KF_Manager`.

6.1 *KF Applet* and Collaborative Applications

To demonstrate the viability of the *KF Applet* concept, we have implemented a shared whiteboard. The changes necessary to turn a stand-alone whiteboard into a shared version are few and straightforward. First, the whiteboard applet was derived from `KF_Applet` instead of `java.awt.applet`. Second, we implemented a set of remote methods to handle AWT events. And finally, we explicitly added RMI calls to propagate AWT events to all partners of a collaborative session. Unlike [18], Knitting-

[2] Note that this is possible since the applet was downloaded from this very machine using, e.g., the *KF Class Server*.

[3] It is of course possible to include security checks, e.g., a password, in this registration call.

Factory does not require re-implementation of the AWT package or a centralized forwarding agent. Also, it is worth emphasizing that the whiteboard applets continue to work even if the `KF_Registry` process is killed—they communicate directly via RMI and do not require a forwarding agent.

7 Security Issues

Web browsers are expected to guarantee the execution of untrusted applet code in a safe manner. To provide this guarantee, various measures are to be used by browsers, including the enforcement of the host-of-origin policy. This policy disallows applets from (1) initiating a network connection to any host other than the one they came from, and (2) listening for network connections initiated elsewhere. It is apparent that the RMI mechanisms used by *KF Applet* violate the host-of-origin policy in providing direct applet-to-applet communication.[4]

The justification for the host-of-origin policy is given as "the intent is to prevent applets from using network connections to circumvent file protections or people's expectations of privacy" [26], another aspect is to protect machines behind a firewall from an attack by an applet downloaded from outside the firewall to a machine within. As discussed earlier, the typical solution [12, 22, 14] to overcome these limitations has been to use a stand-alone application to route messages between applets. *KF Applet* has been designed to only work for applets belonging to the same session. In some sense, this does not violate the spirit of Java, but only the specific technical mechanisms. The usefulness of inter-applet communication has been demonstrated, and it may be possible to formally incorporate certain inter-applet communication in future versions of Java in a secure fashion.

8 Conclusions

We presented KnittingFactory, an infrastructure which supports building Web-based parallel and collaborative applications. Within the context of current research efforts, we pointed out a set of problems and challenges which application programmers face when using Java in combination with Web browsers and their ability to load and execute untrusted applets in a secure fashion. We described the three services provided by KnittingFactory addressing these issues: *KF Directory Service* which makes it possible for users using Web browsers to find other members of a distributed session; *KF Class Server* which allows users to start a distributed computation on any host; and *KF Applet* to allow direct applet-to-applet communication without the need for a forwarding agent. We also argued how parallel programming and collaborative systems can benefit from using KnittingFactory as an underlying layer. We demonstrated this by utilizing KnittingFactory to extend Charlotte, a parallel programming environment, as well as by building a collaborative application.

Currently, KnittingFactory lacks a clean solution to inform applets when a member of the session leaves voluntarily, for example when a user clicks on her browser's `back` button, or involuntarily because of a failure. Future work includes implementing a set of high-level services to support parallel and collaborative applications concurrently, as well as addressing the issue of fault propagation. We are also looking at ways of supporting inter-applet communication in a secure fashion.

Acknowledgements

We are thankful to Ulrich Gall, Li Gong, Alain Mayer, Fabian Monrose, Toto Paxia, Avi Rubin, Peter Wyckoff, Leonid Zheleznyak, and the anonymous referees for helpful comments and for bringing the security concerns to our attention.

This research was sponsored by the Defense Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0320; by the National Science Foundation under grant number CCR-94-11590; and by the Intel Corporation. One of the authors was supported by Deutsche Forschungsgemeinschaft.

The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

[4] Experiments were conducted using Java Developers Kit version 1.1.3, and HotJava Browser Version 1.1 running on Linux and Windows NT 4.0.

References

[1] V. S. Sunderam, A. Geist, J. Dongarra, and R. Mancheck. The pvm concurrent computing system: Evolution, experiences, and trends. *Parallel Computing*, 20:531–546, April 1994.

[2] W. Gropp, Lust E., and Skjellum A. *Using MPI: Portable parallel programming with the message-passing interface*. MIT Press, 1994.

[3] Jon Siegel. *CORBA Fundementals and Programming*. Wiley, 1997.

[4] N. Brown and C. Kindell. *Distributed Component Object Model Protocol - DCOM/1.0*, 1996.

[5] A. Baratloo, M. Karaul, H. Karl, and Z. Kedem. Knittingfactory: An infrastructure for distributed web applications. Technical Report TR 1997-748, Deptartment of Computer Science, New York University, New York, NY, November 1997.

[6] J. E. Baldeschwieler, R. D. Blumofe, and E. A. Brewer. ATLAS: An infrastructure for global computing. In *Proc. of the 7th ACM SIGOPS European Workshop: Systems support for Worldwide Applications*, Connemara, Ireland, September 1996.

[7] A. Baratloo, M. Karaul, Z. Kedem, and P. Wyckoff. Charlotte: Metacomputing on the web. In *Proc. of the 9th International Conference on Parallel and Distributed Computing Systems*, Dijon, France, September 1996.

[8] A. Ferrari. JPVM – the java parallel virtual machine. http://www.cs.virginia. edu/˜ajf2j/jpvm.html.

[9] M. Philippsen and M. Zenger. JavaParty—transparent remote objects in java. In *Proc. of the ACM PPoPP Workshop on Java for Science and Engineering Computation*, Las Vegas, NV, June 1997.

[10] P. Cappello, B. Christiansen, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu. Javelin: Internet-based parallel computing using java. *Concurrency: Practice and Experience*, November 1997.

[11] T. Brecht, H. Sandhu, M. Shan, and J. Talbot. ParaWeb: Towards world-wide supercomputing. In *Proc. of the 7th ACM SIGOPS European Workshop: Systems support for Worldwide Applications*, Connemara, Ireland, September 1996.

[12] D. Bhatia, V. Camuseva, M. Camuseva, G. Fox, W. Furmanski, and G. Premchandran. Web-Flow – a visual programming paradigm for web/java based on coarse grain distributed computing. *Concurrency: Practice and Experience*, March 1997.

[13] G. Fox and W. Furmanski. Towards web/java based high perfomance distributed computing – an evolving virtual machine. In *Proc. of the 5th IEEE Symp. on High Performance Distributed Computing*, 1996.

[14] S. Hummel, T. Ngo, and H. Srinivasan. SPMD programming in java. *Concurrency: Practice and Experience*, March 1997.

[15] I. Foster and S. Tuecke. Enabling technologies for web-based ubiquitous supercomputing. In *Proc. of the 5th IEEE Symp. on High Performance Distributed Computing*, 1996.

[16] K. M. Chandy, A. Rifkin, P. A. G. Sivilotti, J. Mandelson, M. Richardson, W. Tanaka, and L. Weisman. A world-wide distributed system using java and the internet. In *Proc. of the IEEE International Symposium on High Performance Distributed Computing*, Syracuse, NY, August 1996.

[17] P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, and A. Knoche. Redesigning the web: From passive pages to coordinated agents in pagespaces. In *Proc. of the 3rd International Symposium on Autonomous Decentralized Systems*, Berlin, Germany, April 1997.

[18] B. Kvande. The java collaborator toolset, a collaborator platform for the java(tm) environment. Master's thesis, Department of Computer Science, Old Dominion University, August 1996.

[19] R. Guth. Sun's javaspaces is foundation for future distributed systems. IDG News Services, http://www.javaworld.com/javaworld/jw-09-1997 /jw-09-idgns.javaspaces.html, September 1997.

[20] L. Beca, G. Cheng, G. C. Fox, T. Jurga, K. Olszewski, M. Podgorny, P. Sokolwski, and K. Walczak. Web technologies for collaborative visualization and simulation. Technical Report SCCS-786, Northeast Parallel Architectures Center, January 1997.

[21] M. Wahl, T. Howes, and S. Kille. Lightweight directory access protocol (v3). RFC 2251, December 1997.

[22] Sun Micro Systems Inc. Secure computing with java: Now and the future. http://www.javasoft.com/marketing/collateral/security.html.

[23] Sun Micro Systems Inc. Jdk 1.2 beta 2 documentation. http://java.sun.com/products/jdk/1.2/docs/index.html.

[24] A. Wollrath, R. Riggs, and J. Waldo. A distributed object model for the java system. In *USENIX Journal*, Fall 1996.

[25] D. Flanagan. *JavaScript The Definitive Guide*. O'Reilly & Associates, Inc., 1997.

[26] Sun Micro Systems Inc. Frequently asked questions - java security. http://java.sun.com/sfaq/ #summary, 1997.